Salinas–Theory Manual

Garth Reese[*] Manoj K. Bhardwaj[†] Timothy Walsh[‡]

Sandia National Laboratories
Albuquerque, NM 87185-0847

June 9, 2003

[*]Phone: 845-8640
[†]Phone: 844-3041
[‡]Phone: 284-5374

Revision: 1.1

Date: 2003/04/14 16:28:41

Latest Formal Release: 1.0

Abstract

This manual describes the theory behind many of the constructs in Salinas. For a more detailed description of how to use **Salinas** , we refer the reader to *Salinas, User's Notes*.

Many of the constructs in **Salinas** are pulled directly from published material. Where possible, these materials are referenced herein. However, certain functions in **Salinas** are specific to our implementation. We try to be far more complete in those areas.

The theory manual was developed from several sources including general notes, a *programer_notes* manual, the user's notes and of course the material in the open literature.

(this page intentionally blank)

Salinas Theory Manual

Contents

(this page intentionally blank)

1 Solutions

One thing which makes **Salinas** somewhat unique among the many mechanics codes developed at *Sandia National Labs* is that **Salinas** combines a variety of different solution procedures. These range from modal superposition based solutions to nonlinear transient. As described in the *User's Notes* , these solutions can be combined (or chained) in solution cases. This section of the manual describes the theory behind these individual solutions. For details about particular finite elements, see section 2.

1.1 Time integration

In this section we describe a generalized alpha time integration scheme, which is described in Farhat, Crivelli, and Geradin, 'Implicit time integration of a class of constrained hybrid formulations - Part I: Spectral stability theory', CMAME, 125(1995), 71-107. Chung, J., Hulbert, G.M., 'A Time Integration Algorithm for Structural Dynamics with Improved Numerical Dissipation: The Generalized alpha method", J. Applied Mech., Vol.60, pp. 371-375. (ref. 1 and 2).

1.2 Linear transient analysis

The modified equations of motion of the structure are

$$\begin{aligned} M\left[(1-\alpha_m)a_{n+1}+\alpha_m a_n\right] &+ \hat{C}\left[(1-\alpha_f)v_{n+1}+\alpha_f v_n\right]+ \\ K\left[(1-\alpha_f)d_{n+1}+\alpha_f d_n\right] &= F_{n+1+\alpha_f} \end{aligned} \tag{1}$$

where α_f, α_m are the integration parameters for the generalized α method, and $\hat{C} = C + \alpha M + \beta K$. That is, the damping matrix is the sum of the standard damping matrix C plus the proportional damping terms. Also,

$$F_{n+1+\alpha_f} = F((1-\alpha_f)t_{n+1} + \alpha_f t_n) \tag{2}$$

The time integration scheme is defined as follows

$$\begin{aligned} d_{n+1} &= d_n + \Delta t v_n + \frac{\Delta t^2}{2}\left[(1-2\beta_n)a_n + 2\beta_n a_{n+1}\right] \\ v_{n+1} &= v_n + \Delta t\left[(1-\gamma_n)a_n + \gamma_n a_{n+1}\right] \end{aligned} \tag{3}$$

where γ_n, β_n are the integration parameters for the Newmark method.

Substituting these equations into the modified equation of motion, and collecting terms, we obtain

$$\begin{aligned}\left[M(1-\alpha_m)+\hat{C}(1-\alpha_f)\Delta t\gamma_n+K(1-\alpha_f)\Delta t^2\beta_n)\right]a_{n+1} = \\ F_{n+1+\alpha_f}-\hat{C}(1-\alpha_f)\left[v_n+\Delta t(1-\gamma_n)a_n\right] \\ -K(1-\alpha_f)\left[d_n+\Delta t v_n+\frac{\Delta t^2}{2}(1-2\beta_n)a_n\right] \\ -M(\alpha_m a_n)-\alpha_f(\hat{C}v_n+Kd_n)\end{aligned} \tag{4}$$

We note that equation 4 still contains $\hat{C}$, which in turn contains several terms. Thus we substitute $\hat{C}=C+\alpha M+\beta K$ into equation 4 and collect terms on both sides. We then obtain

$$\begin{aligned}\left[M(1-\alpha_m+\alpha(1-\alpha_f)\Delta t\gamma_n)+C(1-\alpha_f)\Delta t\gamma_n\right]a_{n+1} &\quad + \\ K\left[(1-\alpha_f)\Delta t^2\beta_n+\beta(1-\alpha_f)\Delta t\gamma_n\right]a_{n+1} &\quad = \\ F_{n+1+\alpha_f}-C\left[(1-\alpha_f)(v_n+\Delta t(1-\gamma_n)a_n)+\alpha_f v_n\right] &\quad - \\ K(1-\alpha_f)\left[d_n+\Delta t v_n+\frac{\Delta t^2}{2}(1-2\beta_n)a_n\right] &\quad + \\ K\left[\alpha_f(d_n+\beta v_n)+\beta(1-\alpha_f)(v_n+\Delta t(1-\gamma_n)a_n)\right] &\quad - \\ M\left[\alpha_m a_n+\alpha\alpha_f v_n+\alpha(1-\alpha_f)(v_n+\Delta t(1-\gamma_n)a_n)\right] &\end{aligned} \tag{5}$$

1.3 Nonlinear transient analysis

In the case of a nonlinear transient analysis, the equation of motion is

$$\begin{aligned}M\left[(1-\alpha_m)a_{n+1}+\alpha_m a_n\right] &+ \hat{C}\left[(1-\alpha_f)v_{n+1}+\alpha_f v_n\right]+ \\ (1-\alpha_f)F^{int}_{n+1}+\alpha_f F^{int}_n &= F_{n+1+\alpha_f}\end{aligned} \tag{6}$$

where F^{int}_{n+1} and F^{int}_n are the internal forces at the current and previous time steps, respectively.

Using the tangent stiffness method, we replace F^{int}_{n+1} as

$$F^{int}_{n+1}=F^{int}_n+K_t\Delta d \tag{7}$$

where K_t is the tangent stiffness matrix. Also, we use equation 3, which are the same as in the linear case.

Collecting all terms together, we obtain the following equation

$$\begin{aligned}\left[M(1-\alpha_m)+\hat{C}(1-\alpha_f)\Delta t\gamma_n)+K_t(1-\alpha_f)\Delta t^2\beta_n\right]\Delta a = \\ F_{n+1+\alpha_f}-M\left[(1-\alpha_m)\tilde{a}+\alpha_m a_n\right] \\ -\hat{C}\left[(1-\alpha_f)\tilde{v}+\alpha_f v_n\right]-(1-\alpha_f)\tilde{F}^{int}+\alpha_f F_n^{int}\end{aligned} \tag{8}$$

where $\tilde{a}, \tilde{v}$, and $\tilde{F}^{int}$ are the acceleration, velocity, and internal force from the previous Newton iteration. $\Delta a = \tilde{\tilde{a}} - \tilde{a}$, where $\tilde{\tilde{a}}$ is the unknown acceleration from the current Newton iteration. $\tilde{a}$ is computed simply by summing up all of the acceleration changes for all previous Newton iterations and adding the result to a_n. $\tilde{F}^{int}$ is the internal force from the previous Newton iteration, and $\tilde{d}$, $\tilde{v}$ are defined by the update equations

$$\tilde{v} = v_n + \Delta t\left[(1-\gamma_n)a_n + \gamma_n\tilde{a}\right] \tag{9}$$

$$\tilde{d} = d_n + \Delta t v_n + \frac{\Delta t^2}{2}\left[(1-2\beta_n)a_n + 2\beta_n\tilde{a}\right] \tag{10}$$

Once $\tilde{a}, \tilde{v}$, and $\tilde{F}^{int}$ are computed, the right hand side of equation 8 is known, and the tangent stiffness matrix can be computed. Then equation 8 is solved for Δa, and subsequently all quantities are updated again to obtain the current Newton iterate. We note that at the first Newton iterate, $\tilde{a} = a_n$, $\tilde{v} = v_n$, and $\tilde{d} = d_n$. Also, when the right hand side of equation 8 is equal to zero (or less than a prescribed tolerance), convergence is obtained and thus $\tilde{a} = a_{n+1}$, $\tilde{v} = v_{n+1}$, and $\tilde{d} = d_{n+1}$, i.e. the converged values are for the $n+1$ time step.

We note that equation 8 can be written as

$$A\Delta a = res \tag{11}$$

where A is the dynamic matrix, Δa is the change in acceleration from the previous Newton iteration to the current Newton iteration, and res is the residual, i.e. the amount by which the equations of motion (equation 6) are *not* satisfied by the current iterate. This is an intuitive way to view the solution process, since as the residual (right hand side) goes to zero, the change in the Newton iteration will also converge to zero.

1.4 Constraints on integration parameters

In order to achieve second order accuracy and unconditional stability, we need to satisfy the following conditions

$$\alpha_m < \alpha_f <= \frac{1}{2}$$

$$\gamma_n = \frac{1}{2} - \alpha_m + \alpha_f$$
$$\beta_n \geq \frac{1}{4} + \frac{1}{2}(\alpha_f - \alpha_m) \tag{12}$$

Rather than specify all of the above parameters in the input file, the original paper by Hulbert (referenced above) reduces the number of parameters to one. In addition, with this one parameter the others are computed in such a way that maximized high frequency damping and minimized low frequency damping, while still achieving second order accuracy and unconditional stability. Given $0 \leq \rho_{\text{inf}} \leq 1$, the other parameters are computed as

$$\alpha_m = \frac{2\rho_{\text{inf}} - 1}{\rho_{\text{inf}} + 1} \tag{13}$$
$$\alpha_f = \frac{\rho_{\text{inf}}}{\rho_{\text{inf}} + 1} \tag{14}$$
$$\beta_n = 1/4(1 - \alpha_m + \alpha_f)^2 \tag{15}$$
$$\gamma_n = 1/2 - \alpha_m + \alpha_f \tag{16}$$
$$\tag{17}$$

1.5 Stability and numerical dissapation studies

In this section we present a simple example illustrating stability and numerical dissapation properties. The example is a simple cantilever beam made from hex8 elements. The problem is solved on four processors. It is subjected to an impulse-type loading that lasts for a very short time, and then allowed to vibrate freely. The total energy (elastic + kinetic) is plotted versus time. Note that, for the standard Newmark beta algorithm, the total energy should be conserved. The results are shown in Figure 1. The case with no numerical damping and a tight solver tolerance shows the constant energy, as expected. However, when the solver tolerance is decreased, an instability is observed in the solid line. In the final case, the same loose solver tolerance is set, but a small amount of numerical dissapation is introduced through the parameters α_f and β_n. In this case, the solution is stable and the energy decreases in time, as expected.

1.6 Random Vibration

Details of random vibration analysis are included in a number of papers[1]. These few paragraphs document what was implemented.

[1]see for example, reference 3.

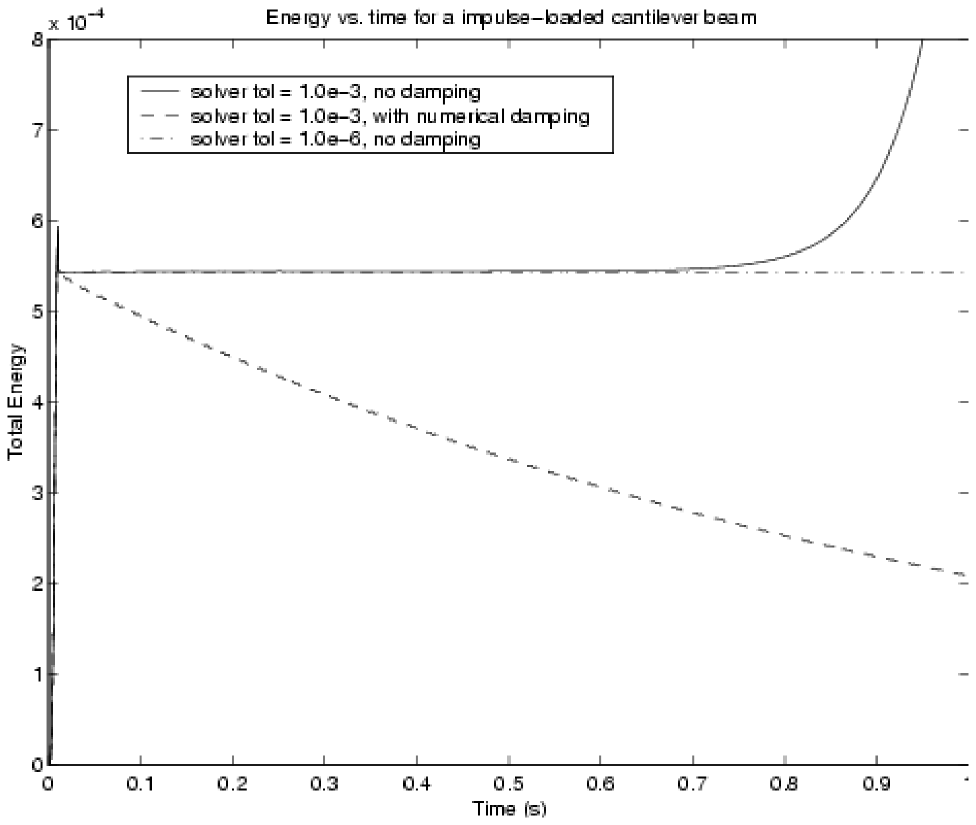

Figure 1: Stability example. A beam of hex8's solved using four processors and varying solver tolerance.

1.6.1 algorithm

The first step in the calculation is computation of Γ_{qq}, which is performed in `ComputeGammaQQ`. This is accomplished as follows.

Let the modal frequency response be defined as,

$$q_i(f) = \frac{1}{\omega_i^2 - \omega^2 + 2j\omega\omega_i\gamma_i}$$

The modal force contribution from load a is,

$$\begin{aligned} F_{i,a}(f) &= \sum_k \phi_{ik} f_k^a s_a(f) \\ &= Z_a^i s_a(f) \end{aligned}$$

where f_k^a is the k component of the force vector associated with load a, and $s_a(f)$ contains all of the frequency content of the force, but none of the spatial dependence. We have defined Z_a^i for each load that represents the sum of all the spatial contributions for mode i. It represents the frequency independent component of the force for load a.

$$Z_a^i = \sum_k f_k^a \phi_{ik}$$

A transfer function to an output degree of freedom, k, from the input load a, may be written as a modal sum.

$$H_{ka}(f) = \sum_i F_{ia}(f) q_i(f) \phi_{ik}$$

where ϕ_{ik} is the eigenvector of mode i.

1.6.2 Power Spectral Density

The displacement power spectral output (at a single location) is a 3×3 matrix.

$$\begin{aligned} G_{mn}(f) &= \sum_{a,a'} H_{ma}^*(f) H_{na'}(f) \\ &= \sum_{i,j} \sum_{a,a'} F_{ia}^*(f) q_i^*(f) \phi_{im} F_{ja'}^*(f) q_j(f) \phi_{jn} \\ &= \sum_{i,j} \sum_{a,a'} q_i^*(f) q_j(f) \phi_{im} \phi_{jn} Z_a^i S^{a,a'}(f) Z_{a'}^j \end{aligned}$$

Here $S^{a,a'}(f)$ is the complex cross-correlation matrix between loads a and a'. The subscripts m and n are applicable to the 3 degrees of freedom at a single location.

By summing over the loads we may reduce the power spectral expression to a sum on modal contributions.

$$G_{mn}(f) = \sum_{i,j} \phi_{im}\phi_{jn}\mathcal{G}_{ij}(f) \tag{18}$$

where

$$\mathcal{G}_{ij}(f) = q_i^*(f)q_j(f)\sum_{a,a'} Z_a^i Z_{a'}^j S^{a,a'}(f) \tag{19}$$

Note that with the exception of the Z_a^i (which may be computed only once and are a fairly small matrix), all the terms in equation 19 are completely known on each subdomain.

1.6.3 RMS Output

The RMS output for degree of freedom m is given by,

$$\begin{aligned} X_{rms} &= \sqrt{\int G_{mm}(f)df} \\ &= \sqrt{\int \sum_{i,j} \phi_{im}\phi_{jm}\mathcal{G}_{ij}(f)df} \\ &= \sqrt{\sum_{i,j} \phi_{im}\phi_{jm}\Gamma_{ij}} \end{aligned}$$

where $\Gamma_{ij} = \int \mathcal{G}_{ij}(f)df$.

The parallel result can be arrived at by computing Z_a^i on each subdomain, and then summing the contributions of each subdomain. Note that Z_a^i contains the spatial contribution of the input force. At boundaries that interface force must be properly normalized just as an applied force is normalized for statics or transient dynamics by dividing by the cardinality of the node. Once Z has been summed, Γ_{ij} may be computed redundantly on each subdomain. The only communication required is the sum on Z (a matrix dimensioned at the number of loads by the number of modes).

The acceleration power spectral density is just $G_{mm}(\omega)\omega^4$. Subsection 2.16.5 provides details about transforming power spectra to an output coordinate system.

1.6.4 RMS Stress

A description of the algorithm for computation of the von Mises RMS stress is included in the reference at the beginning of this chapter. Two methods are available, but both use the integrated modal contribution Γ_{ij} as the basis for their computation. The more complete method relies on a singular value decomposition. Portions of that method are touched on below

1.6.5 matrix properties for RMS stress

Since $S(f)$ is Hermitian, it follows that Γ_{qq} is also necessarily hermitian. It will not in general be real. Therefore, the `svd()` must be computed using complex arithmetic. We use the `zgesvd` routine from `arpack`. The results from the `svd` of an hermitian matrix are real eigenvalues (stored in X), and complex vectors, stored in Q.

At the element level another `svd` must be performed. In this case we are computing the singular values of the matrix C.

$$C = XQ^{\dagger}BQX$$

where,

$$B = \Psi^T A \Psi$$

Obviously, B is symmetric. It can be shown that $Q^{\dagger}BQ$ is hermitian. If we examine a single element of C we can see that it contains the sum over all the terms in an hermitian matrix. That sum is necessarily real, since it can be computed by adding the lower half with it's transpose and then summing the diagonal. Let,

$$A_{ij} = \sum_{m,n} Q^*_{mi} B_{mn} Q_{nj} = \sum_{m,n} a_{ij}$$

But,

$$A^*_{ji} = \sum_{m,n} Qm, j * B_{mn} Q^*_{ni} = \sum_{m,n} Q_{nj} B_{mn} Q^*_{mi} = \sum_{m,n} a^*_{ij}$$

We therefore only need use the real `svd` routines to compute the results at each output location.

1.6.6 model truncation

The `svd` calculations provide the information needed for model truncation. In general, if the size of the model grows, the number of modes required for an analysis also grows. The relationship is very model dependent. However, the computational time for calculating the `svd` varies as the cube of the dimension of the matrix. Since the `svd(Γ)` is only computed once, it is not terribly important. However, the computation of each decomposition of C occurs at each output location and can significantly affect performance. In the model problem where the dimension of C was allowed to remain the same as the number of modes, increasing the number of modes from 20 to 100 changed the time for the analysis by factor of more than 100 (close to the 5^3 one might expect). Clearly, this is unacceptable especially as the desired models may have many hundreds of modes.

The `svd(Γ)` provides important information about the number of independent processes. Note that C includes the `svd` values from this calculation. We truncate by computing all the `nmodes x nmodes` terms in B, but only retaining `Cdim` columns of Q, where `Cdim` is chosen so the values of X are not too small. Thus, $X[(\texttt{Cdim})]/\texttt{X}[0] > \texttt{10}^{-14}$. This restricts the dimension of C to a fairly small number, while retaining all components that contribute significantly to its value. As a result, the entire calculation appears to scale approximately linearly with the number of modes.

1.7 Modal Frequency Response Methods

The Salinas implementation of the modal acceleration method is described in this section. Separate cases are considered when the structure does and does not have rigid body modes.

1.7.1 No Rigid Body Modes

We first consider the frequency domain version of the equations of motion.

$$(-\omega^2 M + j\omega C + K)\hat{u} = \hat{f} \tag{20}$$

Consider the modal approximation

$$\hat{u} \approx \sum_{i=1}^{N} \phi_i q_i \tag{21}$$

where N is the number of retained modes, ϕ_i is the i'th mode shape, and q_i is the i'th modal dof. For modal damping, one obtains the uncoupled equations

$$(-\omega^2 m_i + j\omega c_i + k_i)q_i = \phi_i^T \hat{f} \tag{22}$$

for $i = 1, \ldots, N$ where

$$\begin{aligned} m_i &= \phi_i^T M \phi_i && (23)\\ c_i &= \phi_i^T C \phi_i && (24)\\ k_i &= \phi_i^T K \phi_i && (25)\\ & && (26) \end{aligned}$$

are the modal mass, modal damping, and modal stiffness of the i'th mode. Solving equation 22 for q_i leads to

$$q_i = (\phi_i^T \hat{f})/(-\omega^2 m_i + j\omega c_i + k_i) \tag{27}$$

Replacing $(-\omega^2 M + j\omega C)\hat{u}$ in equation 20 with the modal approximation

$$(-\omega^2 M + j\omega C)\sum_{i=1}^{N} \phi_i q_i \tag{28}$$

leads to

$$K\hat{u} = \hat{f} + (\omega^2 M - j\omega C)\sum_{i=1}^{N} \phi_i q_i \tag{29}$$

Recall that the mode shapes satisfy the eigenproblem

$$K\phi_i = \omega_i^2 M\phi_i \tag{30}$$

where ω_i is the circular frequency of the i'th mode. Provided $\omega_i \neq 0$, one obtains

$$K^{-1}M\phi_i = \phi_i/\omega_i^2 \tag{31}$$

In addition, see Eq. (18.14) of Craig, the damping matrix C can be expressed as

$$C = \sum_{i=1}^{N} \left(\frac{2\zeta_i\omega_i}{m_i}\right)(M\phi_i)(M\phi_i)^T \tag{32}$$

where ζ_i is the damping ratio of the i'th mode. Substituting equations 31 and 32 into equation 29 and solving for $\hat{u}$ leads to

$$\hat{u} = K^{-1}\hat{f} + \sum_{i=1}^{N}(\omega^2/\omega_i^2 - 2\zeta_i j\omega/\omega_i)\phi_i q_i \tag{33}$$

The acceleration frequency response, $\hat{a}$, can be obtained by multiplying equation 33 by $-\omega^2$.

1.7.2 Rigid Body Modes

The procedure outlined here describes how the modal acceleration method can be used in the case when the structure has rigid body modes. The main difference between the approach presented here and Craig's method[4] (pp. 368-371) is in the way that the flexible response is computed using the singular stiffness matrix. Craig removes the rigid body modes from the stiffness matrix using constraints. In our approach, we first orthogonalize the right hand side with respect to the rigid body modes, and then use FETI to solve the singular system directly. Although the two methods are equivalent the latter is much more convenient from the implementation point of view. Note, however, that the implementation is likely to fail on a single processor since the direct solvers are unable to manage a singular stiffness matrix.

The equations of interest are the frequency domain equations of motion

$$-\omega^2 Mu + j\omega Cu + Ku = f \tag{34}$$

Since the stiffness matrix may be singular, we first split the solution into a rigid body part and a flexible part.

$$\begin{aligned} u(\omega) &= u_R(\omega) + u_E(\omega) & (35)\\ &= \Phi_R q_R(\omega) + \Phi_E q_E(\omega) & (36) \end{aligned}$$

where the subscript R refers to rigid body mode contributions, and E refers to contributions from flexible modes. We define N as the total number of degrees of freedom, N_R as the number of rigid body modes and N_E the number of flexible modes, where $N = N_R + N_E$. Then, Φ_R is an NxN_R matrix of rigid body eigenvectors, Φ_E is an NxN_E matrix of flexible eigenvectors, q_R is a vector of dimension N_R, and q_E is a vector of dimension N_E. We assume mass normalized eigenvectors.

We now substitute equation 36 into equation 34, and premultiply both sides by Φ_R^T and Φ_E^T. This yields two sets of equations, after using orthogonality and the fact that $K\Phi_R = 0$.

$$\begin{aligned} -\omega^2 q_R + j\omega C_R q_R &= \Phi_R^T f & (37)\\ -\omega^2 q_E + j\omega C_E q_E + K_E q_E &= \Phi_E^T f & (38) \end{aligned}$$

where C_R, C_E are diagonal matrices containing the modal damping contributions, and K_E is a diagonal matrix containing the eigenvalues. In particular, the ith diagonal entry of C_E is $2\omega_i\zeta_{E_i}$, and the ith diagonal entry of C_R is $2\omega_i\zeta_{R_i}$. For most applications, C_R is null. Solving these equations we obtain the component-wise values of the coefficients

$$q_{R_i} = \frac{\Phi_{R_i}^T f}{-\omega^2 + j\omega C_{R_i}} \tag{39}$$

$$q_{E_i} = \frac{\Phi_{E_i}^T f}{-\omega^2 + j\omega C_{E_i} + \omega_E^2} \tag{40}$$

Equation 38 can be solved for q_E, and substituting this into equation 36, we obtain

$$u = \Phi_R q_R + \Phi_E K_E^{-1} \Phi_E^T f + \omega^2 \Phi_E K_E^{-1} q_E - j\omega \Phi_E K_E^{-1} C_E q_E \tag{41}$$

The first term in equation 41 is known. The third and fourth terms of equation 41 can be computed by modal truncation, and in fact these are the same as the second and third terms of equation 33. The second term in equation 41 is the static correction, and is not readily computable in the present form since all of the flexible modes would have to be known to compute it.

In order to compute the second term in equation 41, we note that the matrix $a_E = \Phi_E K_E^{-1} \Phi_E^T$ is the inverse of the elastic stiffness matrix, that is, the stiffness matrix without the rigid body components. Craig gives a procedure of constraining the rigid body modes in the stiffness matrix in order to compute the product $a_E f$. This procedure would require re-sizing the global stiffness matrix midway through the modalfrf solution procedure, and this is tedius from the code development standpoint.

A more convenient approach is to use FETI to solve the system $Ku = f_E$, where f_E is obtained by orthogonalizing the right hand side f with respect to the rigid body modes, via Gram Schmidt. We note that FETI can solve problems of the form $Ku = f$ even if K is singular, provided that the right hand side f is orthogonal to the rigid body modes.

The procedure is to first apply Gram Schmidt orthogonalization to obtain f_E. Then, we use FETI to solve the system $Ku_E = f_E$, where K is singular. Finally, to be sure u_E is orthogonal to the rigid body modes, we apply Gram Schmidt one more time to u_E. Though in theory u_E is already orthogonal to the rigid body modes after the FETI solve, numerical roundoff may result in a small loss of orthogonality (especially if the solver tolerance is loose), and thus we apply this final orthogonalization to u_E to be on the safe side. The resulting solution we again denote by u_E. Then,

$$u_E = \Phi_E K_E^{-1} \Phi_E^T f \tag{42}$$

and thus all of the terms in equation 41 are known. Thus the modal frequency response can be computed using equation 41.

We note that the orthogonalizations refered to above involve only the standard dot products. That is, in order to make f orthogonal to one rigid body mode ϕ_i, the Gram Schmidt factor is

$$\alpha = \frac{\phi_i^T f}{\phi_i^T \phi_i} \tag{43}$$

and then

$$f_E = f - \alpha\phi \tag{44}$$

The dot products appearing in these expressions do not involve the mass matrix. They are the standard dot products.

1.7.3 Example

Finally, we present an example of the performance of this method as compared to the stndard modal displacement method. The example is a beam composed of 320 hex8 elements. The beam is free-free, so that all rigid body modes are present. The frequency response is computed up to 9000 Hz, and 15 modes are used in the modal expansions. The 15th mode had a frequency of 11362 Hz. In Figure 2, the two methods are compared with the direct frequency response approach. It is seen that the modal acceleration method gives a significantly improved performance over the modal displacement method.

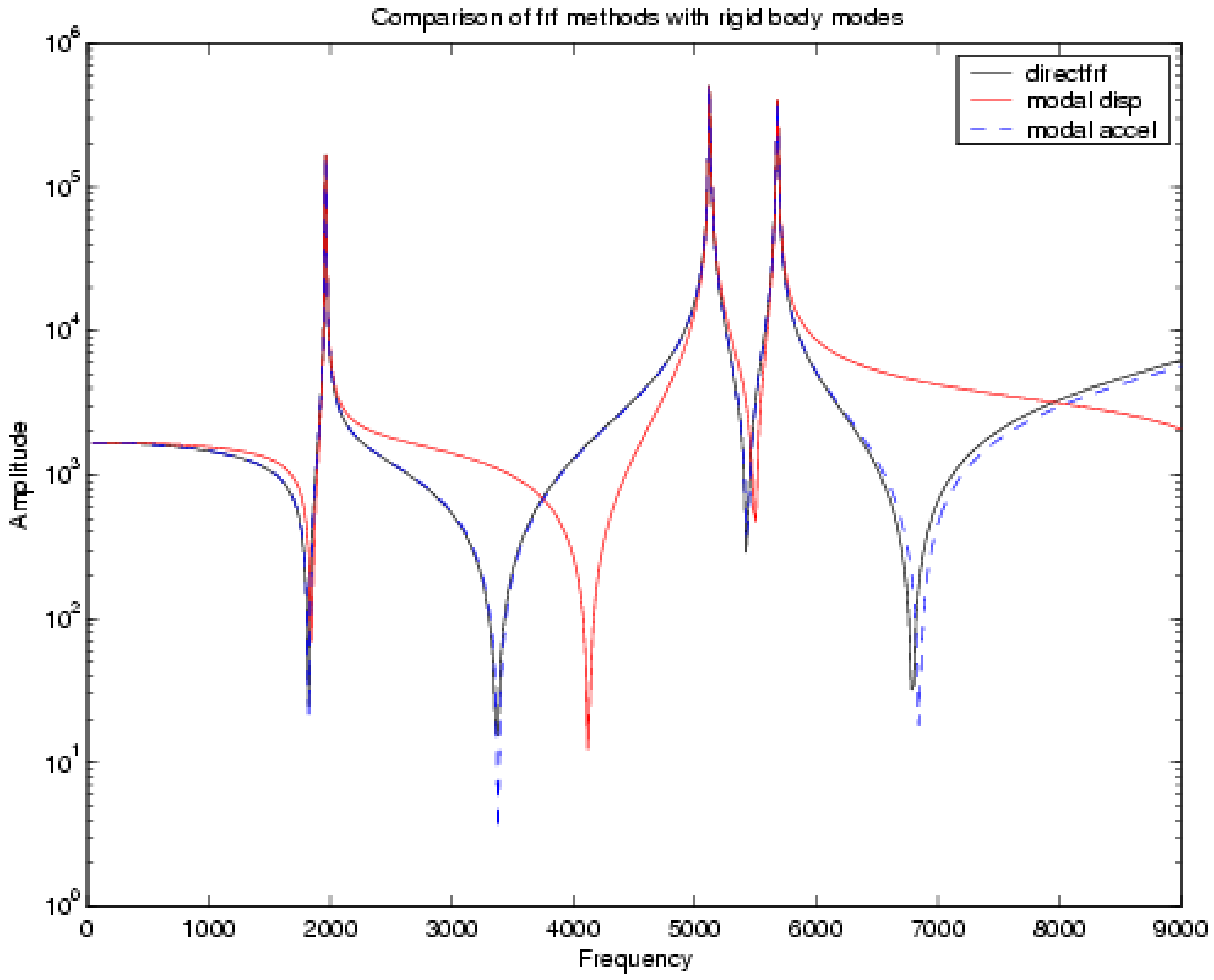

Figure 2: A comparison of the modal displacement, modal acceleration, and direct frequency response approaches. The modal acceleration method gives a better approximation to the direct approach than the modal displacement method.

2 Elements

Structural dynamics is a rich and extensive field. Finite element tools such as **Salinas** have been used for decades to describe and analyze a variety of structures. The same tools are applied to large civil structures (such as bridges and towers), to machines, and to micron sized structures. This has necessarily led to a wealth of different element libraries. Details of these element libraries are presented in this section. For information on the solution procedures that tie these elements together, please refer to section 1.

2.1 Isoparametric Solid Elements. Selective Integration

The following applies to any solid isoparametric element, but is implemented in code on elements with linear shape functions (such as hex8 or wedge6). This discussion addresses calculation of relevant operators on the shape functions and eventual integration into the stiffness matrices. [2]

2.1.1 Derivation

We begin with the separation of the strain into deviatoric and dillitational parts so that their contributions to the stiffness matrix can be computed separately. This is part of the strategy for avoid ing overstiffness with respect to bending.

The strain energy density in the case of an isotropic, linearly elastic material is:

$$p = \frac{1}{2}(2G\epsilon + \lambda tr(\epsilon)I) \bullet \epsilon \tag{45}$$

with some re-arrangement, this can be shown to be:

$$p = G\hat{\epsilon} \bullet \hat{\epsilon} + \frac{1}{2}\beta(tr(\epsilon))^2 \tag{46}$$

where $\hat{\epsilon} = \epsilon - \frac{1}{3}tr(\epsilon)I$.

Having separated the part of the strain energy density due to deviatoric part of the strain from the part of the strain energy density due to the dillitational part of the strain, we shall integrate them separtely. First, we must determine how to express the strains in terms of nodal degrees of freedom.

We know that the deformation field is linear in the nodal degrees of freedom and that the displacement gradient is also, so we should be able to expand each of those quantities as follows.

[2] This development is based on work by Dan Segalman.

Let P_j be the node associated with the $j$the degree of freedom and let s_j be the direction associated with that degree of freedom. The displacement field is:

$$\vec{u}(x) = \tilde{N}^{P_j}(x) u_{s_j}^{P_j} \vec{e}_{s_j} \tag{47}$$

where summation takes place over the degree of freedom j.

Similarly, the displacment gradient is:

$$\vec{\nabla}\vec{u}(x) = (\frac{\partial}{\partial x_k})\tilde{N}^{P_j}(x) u_{s_j}^{P_j} \vec{e}_{s_j} \vec{e}_k \tag{48}$$

We now define the shape deformation tensor W^j corresponding to the j th nodal degee of freedom:

$$W^j(x) = (\frac{\partial}{\partial u_{s_j}^{P_j}})\vec{\nabla}\vec{u}(x) \tag{49}$$

which, with Equation 48 yields:

$$W^j(x) = (\frac{\partial}{\partial x_k})\tilde{N}^{P_j}(x)\vec{e}_{s_j}\vec{e}_k \tag{50}$$

The symmetric part of this tensor is:

$$S^j(x) = \frac{1}{2}(W^j(x) + W^j(x)^T) \tag{51}$$

and the strain tensor is

$$\epsilon(x) = S^j(x) u_{s_j}^{P_j} \tag{52}$$

From the above, we construct the dillatational and deviatoric portions of the strain in terms of the nodal displacement components:

$$tr(\epsilon(x)) = b^j(x) u_{s_j}^{P_j} \tag{53}$$

where

$$b^j(x) = tr(S^j(x)) \tag{54}$$

Similarly,

$$\hat{\epsilon}(x) = \hat{B}^j(x) u_{s_j}^{P_j} \tag{55}$$

where

$$\hat{B}^j(x) = S^j(x) - \frac{1}{3}b^j(x)I \tag{56}$$

The stiffness matrix is evaluated using the consitutive equation (Equation 46) and the following definition:

$$K_{m,n} = \frac{\partial^2}{\partial u_{s_m}^{P_m} \partial u_{s_n}^{P_n}} \int_{volume} p(x) dV(x) \tag{57}$$

This plus our expressions for strain in terms of the nodal degrees of freedom yield us the following expression for element stiffness:

$$K_{m,n} = G \int_{volume} (\hat{B}^m(x))^T \bullet \hat{B}^n(x) dV(x) \tag{58}$$

$$+\beta \int_{volume} b^m(x) b^n(x) dV(x) \tag{59}$$

The issue of selective integration in the elements is discussed in a Framemaker file /home/djsegal/MPP/notes/IsoInt.frm. The formulation discussed there applies to all the isoparametric solid elements.

2.2 Quadratic Isoparametric Solid Elements

Quadratic elements (elements with bilinear or higher order shape functions) such as the Hex20 and Tet10 are naturally soft and do not need to be softened by positive values of G and β (see section 2.1 and the associated Framemaker file IsoInt.frm for definitions of G and β.) Therefore, G=0 and β=0 are good values for such elements.

2.3 Wedge elements

2.3.1 Shape Functions

The shape functions are given explictly in *Hughes*. These are provided as bi-linear polynomials in r, s, t, and ξ, where r and s are independent coordinates of the triangular cross-subsections, $t = 1 - r - s$, and ξ is the coordinate in the third direction. For our purposes, it is necessary to expand the shape functions as polynomials in r, s, and ξ:

$$N_k = A_0^k + A_1^k r + A_2^k s + A_3^k \xi + A_4^k r\xi + A_5^k s\xi \tag{60}$$

The shape functions and the coefficients are given in the following table:

Shape Function	A_0	A_1	A_2	A_3	A_4	A_5
$N_1 = \frac{1}{2}(1-\xi)r$		$\frac{1}{2}$			-$\frac{1}{2}$	
$N_2 = \frac{1}{2}(1-\xi)s$			$\frac{1}{2}$			$-\frac{1}{2}$
$N_3 = \frac{1}{2}(1-\xi)t$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$N_4 = \frac{1}{2}(1+\xi)r$		$\frac{1}{2}$			$\frac{1}{2}$	
$N_5 = \frac{1}{2}(1+\xi)s$			$\frac{1}{2}$			$\frac{1}{2}$
$N_6 = \frac{1}{2}(1+\xi)t$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$	$\frac{1}{2}$	-$\frac{1}{2}$	-$\frac{1}{2}$

2.3.2 Quadrature

Three reasonable quadratures for wedges that come to mind are indicated in the following table:

No. Points	r	s	ξ
1	1/3	1/3	0
2	1/3	1/3	-1/$\sqrt{3}$
	1/3	1/3	1/$\sqrt{3}$
6	1/6	1/6	-1/$\sqrt{3}$
	1/3	1/6	-1/$\sqrt{3}$
	1/6	1/3	-1/$\sqrt{3}$
	1/6	1/6	1/$\sqrt{3}$
	1/3	1/6	1/$\sqrt{3}$
	1/6	1/3	1/$\sqrt{3}$

2.4 Tet10 elements

The 4-point integration is given in *Hughes* (see 5), and the 16-point integration is given in *Jinyun*. It is believed that a higher order integration is needed for the mass matrix than the stiffness matrix and that the reason is that the mass matrix involves higher degree polynomials. (Using 4-point integration to try to estimate the mass matrix of a natural element resulted in a 30 by 30 mass matrix with several zero eigenvalues.)

2.5 Notes on calculating shape functions and their gradients for the Hex20 element

See file Hex20.frm, which is a Framemaker file with a detailed description of how the shape functions and their gradients are calculated for the Hex20 element.

2.6 Anisotropic Elasticity

Anisotripic elasticity requires special care in the rotation of the matrix of matrerial parameters when those parameters are given in some coordinate system other that in which the element matrices are calculated. A derivation of the formulae for rotating those matrices is given in a framemaker file

/home/djsegal/MPP/notes/anisoConst.frm.

2.7 Triangular Shell Element

The triangular shell element (TriaShell) is derived as follows. The bending d.o.f. (w, θ_x, θ_y) and the membrane d.o.f. (u, v, θ_z) are decoupled. The idea is to obtain the membrane response using Allman's triangle and the bending response using the discrete Kirchoff triangular (DKT) element.

2.7.1 Allman's Triangular Element

Using the formulation given in Ref. 6 and replacing $\cos(\gamma_{ij}) = \frac{y_{ji}}{l_{ij}}$ and $\sin(\gamma_{ij}) = \frac{-x_{ji}}{l_{ij}}$, we get

$$u = u_1\psi_1 + u_2\psi_2 + u_3\psi_3 + \frac{1}{2}y_{21}(\omega_2 - \omega_1)\psi_1\psi_2 + \frac{1}{2}y_{32}(\omega_3 - \omega_2)\psi_2\psi_3 + \frac{1}{2}y_{13}(\omega_1 - \omega_3)\psi_3\psi_1 \tag{61}$$

$$v = v_1\psi_1 + v_2\psi_2 + v_3\psi_3 + \frac{1}{2}x_{21}(\omega_2 - \omega_1)\psi_1\psi_2 - \frac{1}{2}x_{32}(\omega_3 - \omega_2)\psi_2\psi_3 - \frac{1}{2}x_{13}(\omega_1 - \omega_3)\psi_3\psi_1 \tag{62}$$

The stiffness and mass matrices ($[K]_{AT}, [M]_{AT}$) are found using general finite element procedures. Unfortunately, a mechanism exists for this element if the deformations are all zero and the rotations are all the same value. Cook *et al.*[7] have a "fix" for this which has been implemented to avoid undesirable low energy modes produced by this mechanism.

2.7.2 Discrete Kirchoff Element

As for the DKT[8] element, things are not so simple. The nine d.o.f. element is obtained by transforming a twelve d.o.f. element with mid-side nodes to a triangle with the nodes at the vertices only. This is obtained as follows. Using Kirchoff theory, the transverse shear is set to zero at the nodes. And the rotation about the normal to the edge is imposed to be linear. Using these constraints, a nine d.o.f. bending element is derived (DKT) using the shape functions for the six-node triangle. Unfortunately, the variation of w over the element cannot be explicitly written. Therefore, the w variation over the element needs to be calculated before the mass matrix can be obtained.

As stated, the equation for w is not explicitly stated over the element in the derivation by Batoz *at al.*. Using a nine d.o.f. element, a complete cubic cannot be written, since 10 quantities would be needed to get a unique polynomial. The strategy taken here is that the stiffness matrix produced using for the DKT element provides reasonable results, and the derivation of the mass matrix is not as critical. So, the equation for w is taken from Ref. 9, as

$$w = \alpha_1\psi_1 + \alpha_2\psi_2 + \alpha_3\psi_3 + \alpha_4\psi_1\psi_2 + \alpha_5\psi_2\psi_3 + \alpha_6\psi_3\psi_1 + \alpha_7{\psi_1}^2\psi_2 + \alpha_8{\psi_2}^2\psi_3 + \alpha_9{\psi_3}^2\psi_1 \tag{63}$$

For the AT and DKT elements, the stiffness and mass matrices are derived with the help of Maple. The consistenet mass matrix is derived using "normal"

DOF	AT/DKT	ABAQUS	AT/DKT!
x	0.000	0.000	0.000
y	0.000	0.000	0.000
z	-1.405 $\times 10^{-2}$	-1.398 $\times 10^{-2}$	-1.398 $\times 10^{-2}$
θ_x	3.337 $\times 10^{-2}$	3.337 $\times 10^{-2}$	3.337 $\times 10^{-2}$
θ_y	3.106 $\times 10^{-2}$	3.089 $\times 10^{-2}$	3.089 $\times 10^{-2}$
θ_z	0.000	0.000	0.000

Table 1: Comparison of deflections at Node 2

finite element procedures. If a lumped mass matrix is requested then the mass matrix terms associated with the translation d.o.f. are found in the "normal" sense. However, mass matrix terms for the rotational d.o.f. are set to $\frac{1}{125}$ of the translation terms.

In summary, the code has been written which uses the AT and DKT element use in combination as a shell element. The stiffness matrices are calculated without complication. The mass matrix for the AT element is also derived without complication. The mass matrix for the DKT element is derived using an incomplete polynomial, but the results obtained should not be effected very much.

2.7.3 Verification and Validation

The AT element is verified by comparing calculated results with the results published by Allman in Ref. 6. The square plate in pure bending and a cantilvered beam with a parabolic tip load are used as verification examples. The mass matrix is not verified except to note that the mass is conserved in the u, v directions.

The DKT element is validated by using the experimental data published by Batoz *et al.* in Ref. 8 for a triangular fin. The first 10 eigenvalues for the triangular fin (cantilever) match very well. In addition, the DKT element is verified by using a cantilevered beam and matching deflection results at the tip. If $\nu = 0$, then results should match very closely with Euler-Beam theory results, and they did.

Finally, the AT/DKT element is verified by comparing with published results from Ref. 10. Tables 1 and 2 show that our elements match exactly with ABAQUS to the number of digits shown. The first column is the result produced by Ertas *et al.*, the second column is the result produced by ABAQUS, and the third column is the result produced by SALINAS using this DKT/AT element.

DOF	AT/DKT	ABAQUS	AT/DKT!
x	0.000	0.000	0.000
y	0.000	0.000	0.000
z	1.949×10^{-2}	1.955×10^{-2}	1.955×10^{-2}
θ_x	3.363×10^{-2}	3.363×10^{-2}	3.363×10^{-2}
θ_y	-2.686×10^{-2}	-2.702×10^{-2}	-2.702×10^{-2}
θ_z	0.000	0.000	0.000

Table 2: Comparison of deflections at Node 3

2.8 Two Node Beam

This is the definition for a Beam element based on Cook's development (see pp 113-115 of reference 7).

The beam uses underintegrated cubic shape functions. Only isotropic material models are supported. Torsional affects are accounted for in the axis of the beam. The beam is uniform in area and bending moments, i.e. they are not a function of position in the beam.

The following parameters are read from the exodus file.

1. The cross subsectional area of the beam (Attribute 1)

2. The orientation of the beam (Attributes 2, 3 and 4)

   The orientation should not be aligned with the beam axis. In the event of an inproperly specified orientation, a warning will be written, and a new orientation selected. The orientation is an x,y,z triplet specifying a direction. It does not need to be completely perpendicular to the beam axis, nor is it required to be normalized. The orientation vector, and the beam axis define the plane for the first bending direction.

3. The first bending moment, I1. (Attribute 5).

4. The second bending moment. I2. (Attribute 6).

5. The torsional moment, J. (Attribute 7).

2.9 Truss

This is the definition for a Truss element based on pages 214-216 of Cook (ref 7).

The truss uses linear shape functions. Unlike the truss elements used by Nastran, there is no torsional stiffness. The truss is uniform in area, i.e. the area is not a function of position in the truss.

The following parameters are read from the exodus file.

1. The cross subsectional area of the truss (Attribute 1)

2.10 Springs

The *Spring* element is the simplest one dimensional element. It has no mass. Entries in the stiffness matrix are added by hand. Note the following.

- The force generated in a *Spring* element should be colinear with the the nodes. Typically spring elements connect coincident nodes so that no torques are generated.
- *Springs* attach 3 degrees of freedom. In the event that some of the spring constants are zero, there is no effective stiffness for that associated degree of freedom. However, the degree of freedom will remain in the A-set matrices. This will be a problem if the other degrees of freedom are not attached to other elements which provide stiffness entries connecting them to the remainder of the model. For an understanding of the various solution spaces (such as the A-set), see section 3.1.

The data for spring elements is entered in the input file. Three values are given, Kx, Ky, and Kz. This results in a 6x6 element stiffness matrix,

$$K' = \begin{pmatrix} K_x & 0 & 0 & -K_x & 0 & 0 \\ 0 & K_y & 0 & 0 & -K_y & 0 \\ 0 & 0 & K_y & 0 & 0 & -K_z \\ -K_x & 0 & 0 & K_x & 0 & 0 \\ 0 & -K_y & 0 & 0 & K_y & 0 \\ 0 & 0 & -K_z & 0 & 0 & K_z \end{pmatrix}$$

Notice that K' is blocked. It could be written more simply,

$$K' = \begin{pmatrix} K'_{11} & K'_{12} \\ K'_{12} & K'_{11} \end{pmatrix}$$

The rotation matrix for the two endpoints is block diagonal. As a result, the stiffness matrix in the basic coordinate system can be written,

$$K = \begin{pmatrix} K_{11} & K_{12} \\ K_{12} & K_{11} \end{pmatrix}$$

where,

$$K_{ij} = R^T K'_{ij} R$$

and R is the 3x3 rotation matrix of subsection 2.15.

2.11 Multi-Point Constraints, MPCs

A description of *MPC*s is contained in the users manual. This subsection discusses the coordinate system dependencies.

*MPC*s may be defined in any coordinate system. However, all nodes in the *MPC*s are defined in the same system. This is done for convenience in parsing, and not for any fundamental reason. Consider a constraint equation where each entry in the equation could be specified in a different coordinate system.

$$\sum_i C_i u_i^{(k_i)} = 0$$

where C_i is a real coeffient, and $u_i^{(k_i)}$ represents the displacement of degree of freedom i in degree of coordinate system k_i. We can transform to the basic coordinate system using $u_i^{(k_i)} = \sum_j R_{ji}^{(k_i)} u_j^{(0)}$, where $R^{(k_i)}$ is the rotation matrix for coordinate system k_i. Then we may write,

$$\sum_{i,j} C_i R_{ji}^{(k_i)} u_j^{(0)} = 0$$

or,

$$\sum_i C_i^{(k_i)} u_i^{(0)} = 0$$

where $C_i^{(k_i)} = \sum_j R_{ij}^{(k_i)} C_j$. Note however, that in this analysis, we have assumed that the dimension of C is 3. Thus, rotation into the basic frame will likely increase the number of coefficients.

Salinas is designed to support constraints through at least two methods. This include a constraint transform method and Lagrange multipliers. Lagrange multipliers have not been implemented at this time.

2.11.1 Constraint Transforms

Constraints may be eliminated using the constraint transform method. This is described in detail in Cook, chapter 9 (ref 7). In this method, the analysis set is partitioned into constrained degrees of freedom and retained degrees of freedom. The constrained dofs are eliminated.

Unlike many Finite Element programs, Salinas does not support user specification of constraint and residual degrees of freedom. The partition of constrained and retained degrees of freedom is performed simultaneously in the “gauss()” routine. This routine performs full pivoting so the constrained degrees of freedom are guaranteed to be independent. Redundant specification of constraint equations is handled by elimination of the redundant equations and issue of a warning. User selection of constrained dofs in Nastran has led to serious difficulty to insure that the constrained dofs are independent and never specified more than once.

For constraint elimination we have a constraint matrix $C = C_c C_r$, where C_c is a square, nonsingular matrix and C_r is the solution. We wish to solve for,

$$C_{rc} = -[C_c]^{-1} C_r$$

This is equivalent to the Gauss-Jordan elimination probrlem for $Kx = b$ if we let $C_r = b$, $C_c = K$ and $x = -C_{rc}$. There is one additonal wrinkle: we have mixed the rows of C so C_c is intermingled with C_r. However, we only require that C_C be non-singular. Therefore if we do a gauss elimination with full pivoting we should simultaneously obtain an acceptable reordering of C, and botain C_{rc}.

In practice, it is not even necessary that C_c be non-singular. It is not uncommon for two identical constraints to be specified. The program issues a warning and continues.

Constraint transform methods do not currently support recovery of MPC forces.

The Gaussian elimination is presently being performed with a sparse package called ”SuperLU,” instead of a dense gaussian elimination, to speed up the time to create C_{rc}. On some platforms, e.g., sgi and dec, the blas routine dmyblas2.c in the CBLAS directory of of the SuperLU directory (need superlu-underscore-salinas.tar to create this) should be the one and only routine whose object file is placed into the SuperLU-blas library (presently called libblas-underscore-super.a) to be linked in to create the salinas executable. Failure to include this routine will cause failures of the type ”Illegal value in call to DSTRV” on the above platforms, and including more than just dmyblas2.c can cause slow performance on many platforms as the SuperLU-CBLAS could override the built-in blas routines. (The built-in routines are almost always faster.)

2.12 Rigid Elements

Salinas supports standard *pseudo*elements for rigid bodies. These include,

- RRODs - a rigid truss like element, infinitely stiff in extension, but with no coupling to bending degrees of freedom.

- RBARS - a rigid beam, 6 degrees of freedom deleted

- RBE2 - a rigid solid. $6(n-1)$ degrees of freedom deleted, where n is the number of nodes
- RBE3 - an averaging type solid. This connects to many nodes, but removes only 6 dofs.

All of the rigid elements are stored and applied internally as MPC equations. The RBE2 is a special case of RBAR (actually just multiple instances). Note, that unlike MPC equations, these rigid elements do activate (or touch) degrees of freedom. In general, an MPC equation will not activate a degree of freedom. In the case of a rigid element however, it is necessary to activate the degrees of freedom before constraining them. Otherwise the rigid elements do not act like real elements.

Rigid elements are input into Salinas using exodus beam elements. A block entry is then provided in the input file indicating what type of rigid element is required. There is no stiffness or mass matrix entry for any type of rigid elements (only the MPC entries described above).

2.12.1 RROD

An RROD is a *pseudo*element which is infinitely stiff in the extension direction. The constraints for an RROD may be conveniently stated that the dot product of the translation and the beam axial direction for a RROD is zero. There is one constraint equation per RROD.

2.12.2 RBAR

An RBAR is a *pseudo*element which is infinitely stiff in all the directions. The constraints for an RBAR may be summarized as follows.

1. the rotations at either end of the RBAR are identical,
2. there is no extension of the bar, and
3. translations at one end of the bar are consistent with rotations.

It is apparent that the last two of these constraints may be specified mathematically by requiring that the translation be the cross product of the rotation vector and the bar direction.

$$\vec{T} = \vec{R} \times \vec{L}$$

where $\vec{T}$ is the translation difference of the bar (defined as $\vec{U}_2 - \vec{U}_1$),
$\vec{R}$ is the rotation vector, and
$\vec{L}$ is the vector from the first grid to the second.

The three constraints in the cross product, together with the three constraints requiring identical rotations at both ends of the bar form the six required constraint equations.

2.12.3 RBE3

The RBE3 elements behavior is taken from Nastran's element of the same name. Note however, that the precise mathematical framework of the Nastran RBE3 element is not specified in the open literature. This element should act like an RBE3 for most applications. The element is used to apply distributed forces to many nodes while not stiffening the structure as an RBE2 or RBAR would. The RBE3 uses the concept of a slave node. Constraints are specified as follows.

1. The translation of the slave node is the sum of translations of all the other nodes in the element.

2. The rotation of the slave node is the weighted average rotation of all the other nodes about it.

While the first of these constraints is easy enough to apply using multi-point constraints, the second is a little more difficult. We seek a least squares type solution.

Let $\vec{D}_i = \vec{U}_i - \vec{U}_{slave}$,
$\vec{L}_i = \vec{X}_i - \vec{X}_{slave}$

The L represent a vector from the "origin" to the point i, while the D_i represent the differential displacement of the same points. Note that the origin is at the location of the slave node, and will not in general be at the centroid of the structure.

We will use least squares to compute the rotational vector of the slave node. This is equivalent to computing a rotational inertial term and requiring a similar net rotation for the centroid.

The displacement at the centroid should be given by,

$$\vec{D}_i = \vec{R} \times \vec{L}_i$$

or, in the least squares sense we seek to minimize E.

$$E = \sum_i (\vec{D}_i - \vec{R} \times \vec{L}_i) \cdot (\vec{D}_i - \vec{R} \times \vec{L}_i)$$

Take the derivative of E with respect to a component of R, r_k.

$$\frac{dE}{dr_k} = 0 = 2 \sum_i (\hat{e}_k \times \vec{L}_i) \cdot (\vec{R} \times \vec{L}_i) - \vec{D}_i \cdot (\hat{e}_k \times \vec{L}_i)$$

Now, let $R = \sum_m r_m \hat{e}_m$. We substitute for R in the previous equation to obtain,

$$\sum_m \sum_i r_m(\hat{e}_k \times \vec{L}_i) \cdot (\hat{e}_m \times \vec{L}_i) - \vec{D}_i \cdot (\hat{e}_k \times \vec{L}_i) = 0$$

Now, if we write L_i as a column vector then the expression $(\hat{e}_k \times \vec{L}_i) \cdot (\hat{e}_m \times \vec{L}_i)$ can be written as $L_i^T L_i \cdot I - L_i L_i^T$. If the sum on i is performed for the first term, we may write,

$$\sum_m r_m A_{mk} - \sum_i \hat{e}_k \cdot (\vec{L}_i \times \vec{D}_i) = 0$$

This provides three equations (one for each k) in the 3 unknowns, r_m.

The solution is found by looping once through all i to fill in the A matrix, and simultaneously to fill out the coefficients for the three equations involving D_i. Once the loop has been completed, the coefficients of R are known, and the three components of r_m can be added for each of the three equations. Each equation has 3 components of R, $2n$ components of U_i and 2 components of U_{slave} for a total of $2n + 5$ equations.

2.13 Shell Offset

Consider a shell offset, with an offset vector, $\vec{v}$. Notice that $\vec{v}$ could be defined at each nodal location in what follows, but for this development, we assume a single offset $\vec{v}$ which applies to all nodes. Define a coordinate system at the node, with variables u. On the offset beam the coordinate system is $\tilde{u}$.

Now, u is related simply to $\tilde{u}$. The constraint of a constant offset may be stated that the displacement difference of the two systems must be orthogonal to $\vec{v}$, i.e. $(u - \tilde{u}) = \vec{v} \times \vec{\kappa}$, where $\vec{\kappa}$ is the rotation at the nodes. Notice that the rotation is the same at both nodes.

Thus we can write,

$$\begin{pmatrix} \tilde{u} \\ \kappa \end{pmatrix} = [L] \begin{pmatrix} u \\ \kappa \end{pmatrix} \tag{64}$$

where L is a constant matrix which depends only on the geometry. We can use this transformation matrix to eliminate the degrees of freedom associated with $\tilde{u}$. The energy of the shell can be written,

$$E_{strain} = 0.5 \begin{Bmatrix} \tilde{u} \\ \kappa \end{Bmatrix}^T \left[\tilde{K}\right] \begin{Bmatrix} \tilde{u} \\ \kappa \end{Bmatrix} \tag{65}$$

But with this substitution,

$$E_{strain} = 0.5 \begin{Bmatrix} u \\ \kappa \end{Bmatrix}^T \left[L^T \tilde{K} L\right] \begin{Bmatrix} u \\ \kappa \end{Bmatrix} \tag{66}$$

If we let $K = L^T \tilde{K} L$, then,

$$E_{strain} = 0.5 \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\}^T [K] \left\{ \begin{array}{c} u \\ \kappa \end{array} \right\} \tag{67}$$

Thus, $\tilde{u}$ has been eliminated, and the equations may be rather simply put in terms of the output variables.

2.14 Notes on Consistent Loads Calculations

Starting with equation 4.1-6 from *Concepts and Applications of Finite Element Analysis* by Cook *et al.*,

$$\{r_e\} = \int_{V_e} [B]^T [E] \{\epsilon_0\} dV - \int_{V_e} [B]^T \{\sigma_0\} dV + \int_{V_e} [N]^T \{F\} dV + \int_{S_e} [N]^T \{\Phi\} dS \tag{68}$$

where each of these terms are defined in Subsection 4.1 of the above mentioned reference. The load vector, $\{r_e\}$, is composed of four parts in Eqn. 68. In this document, only the last part, which is the contribution of the surface tractions to the load vector, will be considered. Rewritting,

$$\{r_e\} = \int_{S_e} [N]^T \{\Phi\} dS \tag{69}$$

Here, the integral is calculated over the surface of the element on which the surface traction, $\{\Phi\}$, is applied. Therefore,

$$\{\Phi\} = [\Phi_x \Phi_y \Phi_z]^T \tag{70}$$

and $[N]$ is the shape function matrix of the element on which the surface tractions, $\{\Phi\}$, are applied. In Salinas, $\{\Phi\}$ can be applied within PATRAN by applying a spatial field to a specified side set. As a result, when calculating the load vector, this field must be accounted for. In Salinas however, this spatial field values will be available only at the nodes of the element. Using the nodal values of this surface traction, the value inside must be defined using an interpolation function over the surface or side of the element. Since only one value per node may be specified on the side set in Salinas, a surface traction may be applied only in one direction at a time. Therefore, $\{\Phi\}$ will be defined as

$$\{\Phi\} = \left\{ \begin{array}{c} n_x \\ n_y \\ n_z \end{array} \right\} \Phi(x, y, z) \tag{71}$$

2.14.1 Salinas Element Types

The following 3-D and 2-D elements have consistent loads implemented:

- Hex8
- Hex20
- Wedge6
- Tet4
- Tet10
- Tria3
- TriaShell
- Tria6 (four Tria3s)
- QuadT (twor Tria3s)
- Quad8T (1 QuadT and 4 Tria3s)

2.14.2 Pressure Loading

Here, we will consider only pressure loads on 3-D elements, such that

$$\{\Phi\} = \left\{ \begin{array}{c} N_x \\ N_y \\ N_z \end{array} \right\} \Phi(x,y,z) \tag{72}$$

where $[N_x, N_y, N_z]^T$ is the normal to the element face. Hence, the consistent loads can be calculated as,

$$\{r_e\} = \int_{S_e} [N]^T \{\Phi\} dS = \int_{S_e} [N]^T \Phi(x,y,z)(\vec{a} \times \vec{b}) dS_e \tag{73}$$

Here,

$$\vec{a} = [\frac{\partial x}{\partial r}, \frac{\partial y}{\partial r}, \frac{\partial z}{\partial r}]^T \tag{74}$$

$$\vec{b} = [\frac{\partial x}{\partial s}, \frac{\partial y}{\partial s}, \frac{\partial z}{\partial s}]^T \tag{75}$$

where Φ is the pressure load, and (x,y,z) are the physical coordinate directions, and (r,s) are the local element directions for the face of the element. Notice, taking the cross-product of $\vec{a}$ and $\vec{b}$, the normal is obtained.

2.14.3 Shape Functions for Calculating Consistent Loads

For 3-D elements, all the faces are either quadrilateral or triangular shaped. Hence, shape functions for quads and triangles could be used to evaluate the consistent loads. If the shape functions for the 3-D elements are used, it will reduce code and "fit" better into the current finite element class structure. This is what is currently implemented. This requires a "mapping" of the 3-D elements' faces to a 2-D plane. The additional overhead for using the 3-D elements is that each face of the element must have this "mapping" which states how the elements' 3-D shape functions will map to a 2-D element. For example, for a Hex20, the element coordiantes (η_1, η_2, η_3) are defined in a particular way. For each face of the Hex20, defined by a side id, the face will have a local coordinate system (r, s). The "mapping" will define how (r, s) are related to (eta_1, eta_2, eta_3). This will also help defined what how 2-D Gauss points are mapped to the 3-D face. These mappings are done for all the 3-D elements.

2.14.4 Shell Elements - consistent loads

All the 2-D elements (shell elements) are based on the Tria3. The consistent loads calculations for the Tria3 can be "copied" to the TriaShell. This way all the shell elements will use the same consistent loads implementation. Since Carlos Felippa designed the Tria3, his consistent loads implementation is used. The portion for linearly varying pressure loads is shown here. If the loads are aligned along an edge, $\{q\}$, they need to be decomposed into (qs, qn, qt). Where (s, n, t) are coordinate directions along the element edge. Coordinate s varies along the element edge tangentially, n is normal to the element edge, and t is tangent to the element edge in the transverse direction, i.e., in the direction of the thickness. Once, the edge load is decomposed, the equations for consistent loads are

$$f^1{}_s = \frac{1}{20}(7q_{s1} + 3q_{s2})L_{21} \qquad f^2{}_s = \frac{1}{20}(3q_{s1} + 7q_{s2})L_{21} \tag{76}$$

$$f^1{}_n = \frac{1}{20}(7q_{n1} + 3q_{n2})L_{21} \qquad f^2{}_n = \frac{1}{20}(3q_{n1} + 7q_{n2})L_{21} \tag{77}$$

$$f^1{}_t = \frac{1}{20}(7q_{t1} + 3q_{t2})L_{21} \qquad f^2{}_t = \frac{1}{20}(3q_{t1} + 7q_{t2})L_{21} \tag{78}$$

$$m^1{}_s = m^2{}_s = 0 \tag{79}$$

$$m^1{}_n = -\frac{1}{60}(3q_{t1} + 2q_{t2})L^2{}_{21} \qquad m^2{}_n = \frac{1}{60}(2q_{t1} + 3q_{t2})L^2{}_{21} \tag{80}$$

$$m^1{}_t = -\frac{1}{40}(3q_{n1} + 2q_{n2})L^2{}_{21} \qquad m^2{}_t = \frac{1}{40}(2q_{n1} + 3q_{n2})L^2{}_{21} \tag{81}$$

where q_{s1} is the value of q in the s direction at node 1 of the edge, L_{12} is the length of the edge. The superscipts 1,2 are the node numbers of the edge. Note, it is assumed here that the load q is per unit length, but this is not assumed when creating the sideset in PATRAN for example. Therefore, this distributed load is multiplied, in Salinas, by the thickness of the triangle.
Now if the pressure load is on the face of the Tria3, the equations become,

$$f^1{}_x = f^1{}_y = m^1{}_z = f^2{}_x = f^2{}_y = m^2{}_z = f^3{}_x = f^3{}_y = m^3{}_z = 0 \tag{82}$$
$$f^1{}_z = (\frac{8}{45}p_1 + \frac{7}{90}p_2 + \frac{7}{90}p_3)A \tag{83}$$
$$f^2{}_z = (\frac{7}{90}p_1 + \frac{8}{45}p_2 + \frac{7}{90}p_3)A \tag{84}$$
$$f^3{}_z = (\frac{7}{90}p_1 + \frac{7}{90}p_2 + \frac{8}{45}p_3)A \tag{85}$$
$$m^1{}_x = \frac{A}{360}[7(y_{31} + y_{21})p_1 + (3y_{31} + 5y_{21})p_2 + (5y_{31} + 3y_{21})p_3] \tag{86}$$
$$m^1{}_y = \frac{A}{360}[7(x_{13} + x_{12})p_1 + (3x_{13} + 5x_{12})p_2 + (5x_{13} + 3x_{12})p_3] \tag{87}$$
$$m^2{}_x = \frac{A}{360}[(5y_{12} + 3y_{32})p_1 + 7(y_{12} + y_{32})p_2 + (3y_{12} + 5y_{32})p_3] \tag{88}$$
$$m^2{}_y = \frac{A}{360}[(5x_{21} + 3x_{23})p_1 + 7(x_{21} + x_{23})p_2 + (3x_{21} + 5x_{23})p_3] \tag{89}$$
$$m^3{}_x = \frac{A}{360}[(3y_{23} + 5y_{13})p_1 + (5y_{23} + 3y_{13})p_2 + 7(y_{23} + y_{13})p_3] \tag{90}$$
$$m^3{}_x = \frac{A}{360}[(3x_{32} + 5x_{31})p_1 + (5x_{32} + 3x_{31})p_2 + 7(x_{32} + x_{31})p_3] \tag{91}$$

where $y_{ij} = y_i - y_j$ and $x_{ij} = x_i - x_j$, A is the area of the triangle, p_i is the value of the pressure load at node i, and (x_i, y_i) are coordinates of the triangle in 2-D space.
Finally, the "pseudo" elements (QuadT, Quad8T, Tria6) created by using Tria3s require a little extra overhead. For example, the Quad8T is composed of 1 QuadT and 4 Tria3s. However, since it is defined as a Quad8T, it will have distribution factors at its 8 nodes, and these distribution factors have to be mapped to the 1 QuadT and the 4 Tria3s. The number of distribution factors will be 3 however, if the load is applied to its edge. Therefore, this extra coding can be seen in the ElemLoad method of the shells' classes.

2.15 Coordinate Systems

Coordinate systems are provided for a number of applications including:

1. specification of boundary constraints (SPCs)
2. specification of multi-point constraints (MPCs)
3. specification of material property rotations for anisotropic materials.
4. specification of spring directions (see subsection 2.10).
5. specification of output coordinate systems (in history files only).

There are some applications for coordinate systems which we do NOT intend to support. These include,

1. specification of nodal locations,
2. specification of new coordinate systems in any but the basic system.

Coordinate systems for cartesian, cylindrical and spherical coordinates may be defined. In the case of noncartesian systems, the XZ plane is used for defining the origin of the θ direction only.

Each coordinate system carries with it a rotation matrix. It is important to clarify the meaning of that matrix. Specifically,

$$X' = RX$$

Where X' is the new system of coordinates, R is the rotation matrix and X is the basic coordinate system. For cartesian systems, the rotation matrix is static. Curvilinear systems will require computation of a new rotation matrix at each location in space.

The usual identity on rotation matrices applies, namely:

$$X = R^T X' \tag{92}$$

and

$$R^T R = R R^T = I$$

As an example, consider a cartesian system as shown in Figure 3.

The new system (marked by primes) is rotated θ from the old system with the new X' axis in the first quadrant of the old system. The rotation matrix is,

$$R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

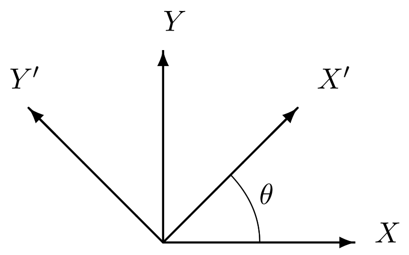

Figure 3: Original, and rotated coordinate frames

2.16 Constraint Transformations in General Coordinate Systems

In general, constraint equations can be applied in any coordinate system. We here describe the transformation equations and implications for general constraints in any coordinate system. The implications of this use in Salinas are also outlined.

Consider a constraint equation,

$$C'u' = Q \tag{93}$$

where the primes indicate a generalized coordinate frame. The frame may be transformed to the basic coordinate system using equation 92, and

$$u' = Ru \tag{94}$$

We can now rewrite equation 93,

$$\begin{aligned} C'Ru &= Q \\ Cu &= Q \end{aligned} \tag{95}$$

where $C = C'R$.

Thus a general system of constraint equations may be easily transformed to the basic system. Further, the rotational matrix is a 3x3 matrix which may be applied to each node's degrees of freedom separately.

2.16.1 Decoupling Constraint Equations

We still have a coupled system of equations. We partition the space into constrained and retained degrees of freedom, and describe the constrained dofs in terms of its Schurr complement.

$$u = \left[\begin{array}{c} u_r \\ u_c \end{array} \right] \tag{96}$$

The whole constraint equation may be similarly partitioned.

$$\begin{bmatrix} C_r & C_c \end{bmatrix} \begin{bmatrix} u_r \\ u_c \end{bmatrix} = [Q] \tag{97}$$

Note that C_r is an cxr matrix, C_c is cxc, and Q is a vector of length c. Under most conditions Q is null.

This may be solved for u_c,

$$u_c = C_c^{-1} Q - C_c^{-1} C_r u_r \tag{98}$$

We must be concerned with cases where C_c may be either singular or over constrained. The former case occurs if we try to eliminate c equations, but the rank of C is less than c. This could occur if the equations are redundant. We can over constrain the system only if Q is nonzero. Both these situations need attention, but both can be dealt with.

We may also write the solution using a transformation matrix, T.

$$\begin{bmatrix} u_r \\ u_c \end{bmatrix} = [T]\,[u_r] + \tilde{Q} \tag{99}$$

where

$$T = \begin{bmatrix} 1 \\ C_{rc} \end{bmatrix} \tag{100}$$

$$C_{rc} = -C_c^{-1} C_r \tag{101}$$

and

$$\tilde{Q} = \begin{bmatrix} 0 \\ C_c^{-1} Q \end{bmatrix} = \begin{bmatrix} 0 \\ \breve{Q} \end{bmatrix} \tag{102}$$

2.16.2 Transformation of Stiffness Matrix

We assume a similar partition of the stiffness matrix. The equations for statics are then,

$$\begin{bmatrix} K_{rr} & K_{rc} \\ K_{cr} & K_{cc} \end{bmatrix} \begin{bmatrix} u_r \\ u_c \end{bmatrix} = \begin{bmatrix} R_r \\ R_c \end{bmatrix} \tag{103}$$

or,

$$[K]\,[T]\,u_r + [K]\left[\tilde{Q}\right] = R \tag{104}$$

and

$$T^T K T u_r = T^T \left\{ R - K\tilde{Q} \right\} = \tilde{R} \tag{105}$$

We can define the reduced equations,

$$\tilde{K} = T^T K T = K_{rr} + K_{rc}C_{rc} + C_{rc}^T K_{cr} + C_{rc}^T K_{cc} C_{rc} \tag{106}$$

and,

$$\begin{aligned} \tilde{R} &= T^T R - T^T \left[\begin{array}{c} K_{rc}\breve{Q} \\ K_{cc}\breve{Q} \end{array} \right] \\ &= R_r + C_{rc}^T R_c - K_{rc}\breve{Q} - C_{rc}^T K_{cc}\breve{Q} \end{aligned} \tag{107}$$

The solution in the retained system is

$$\tilde{K} u_r = \tilde{R} \tag{108}$$

The system may now be solved using the reduced equations, and the constrained degrees of freedom may be solved using equation 98. Much of this is detailed in Cook, but without the constrained right hand side.

For eigen analysis the mass matrix may be transformed exactly as the stiffness matrix in equation 106. There is no force vector.

For transient dynamics the mass and stiffness matrix transform the same. The force vector and force vector corrections may be time dependent. There is currently no structure to store these time dependent terms in Salinas.

2.16.3 Application to single point constraints

Our initial efforts at applying single point constraints (SPC) has been limited to the basic coordinate system. In that system the equations decouple, C_c is unity and C_{rc} is zero. Then equations 106 and 107 reduce to elimination of rows and columns.

To properly account for the coupling that occurs when the constraints are not applied in the basic coordinate system, we must generate all the constraint equation on the node. This may be up to a 6x6 system. I believe that there is no real conflict in first applying constraints in the basic system, then adding additional constraints in other systems.

The process for applying constraints can be summarized as follows.

1. Generate the constraint equation in the generalized coordinate system (equation 93).

2. Transform the constraint equation to the basic coordinate system (equation 94).

3. Determine the constraint degrees of freedom. It may need to be done in concert with the next step to keep from degrading the matrix condition.

4. Compute the two transformation matrices C_c^{-1} and C_{rc} from equations 97 and 101.

5. Compute the corrections to the force vector from equation 107. We currently do not have a structure to store these corrections, except for the case of statics.

6. Compute the reduced mass and stiffness matrices from equation 106.

7. Eliminate the constraint degrees of freedom from the mass and stiffness matrix.

   In addition, for post processing,

8. store the terms of the equations necessary to recover the constraint degrees of freedom (equation 98).

A few words about post processing could also prove useful. In the first implementation of Salinas, constraints were applied only in the basic coordinate system. The degree of freedom to eliminate was obvious from the exodus file, and it's value was a constant (usually zero). In this later version, a more general approach must be used. We use the following strategy.

1. degrees of freedom directly constrained to zero are handled implicitly. This is done by setting the G-set vector to zero before merging in the A-set results. There is no storage cost for this.

2. Other degrees of freedom are managed using an spc_info object. An array of these objects will be stored globally. Each object contains the degree of freedom to fill, an integer indicating the number of other terms, a list of dofs/coefficients, and a constant. This facilitates solutions of the form,

$$u_{\text{spc}} = \text{constant} + \sum_{i}^{\text{retained dofs}} u_i C_i \tag{109}$$

2.16.4 Multi Point Constraints

The application to multipoint constraints is very straight forward. The only difference is that the whole system of equations must be considered together. This changes the linear algebra significantly because the matrices must now be stored in sparse format. However, the steps that are applicable for single point constraints apply here as well. Subsection 2.11 deals more explicitly with MPCs.

2.16.5 Transformation of Power Spectral Densities

Note: The following is taken almost verbatim from Paez's book. We identify how to transform output PDS.

Let $\mathbf{H}(f)$ denote a frequency response function vector for a given input (in the global system) expressed as,

$$\mathbf{H}(f) = H_1(f)\mathbf{e}_1 + H_2(f)\mathbf{e}_2 + H_3(f)\mathbf{e}_3$$

where $\mathbf{e}_i$ represents the unit vectors of this space. Note that $\mathbf{H}(f)$ is an output vector at a single location in the model. $\mathbf{H}(f)$ can also be expressed using an alternate set of unit vectors, $\tilde{\mathbf{e}}_i$.

$$\mathbf{H}(f) = \tilde{H}_1(f)\tilde{\mathbf{e}}_1 + \tilde{H}_2(f)\tilde{\mathbf{e}}_2 + \tilde{H}_3(f)\tilde{\mathbf{e}}_3$$

Taking the dot product of these two equations and equating the results, we have,

$$\tilde{H}_1(f) = \sum_{k=1}^{3} c_{ki} H_k(f) \tag{110}$$

where

$$c_{ki} = \mathbf{e}_k \cdot \tilde{\mathbf{e}}_i$$

The spectral density function $G_{ij}(f)$ (for a given input and at a single output location) can be expressed as,

$$G_{ij}(f) = \alpha H_i^*(f) H_j(f) \tag{111}$$

where α is a constant and superscript * denotes complex conjugate. Similarly for the alternative coordinate frame,

$$\tilde{G}_{ij}(f) = \alpha \tilde{H}_i^*(f) \tilde{H}_j(f)$$

We may use equation 110 to express $\tilde{G}$ in terms of the H_i. We may then use the spectral definition in equation 111 to provide the transformation of spectral densities.

$$\begin{aligned} \tilde{G}_{ij}(f) &= \alpha \left(\sum_{k=1}^{3} c_{ki} H_k^*(f) \right) \left(\sum_{m=1}^{3} c_{mj} H_m(f) \right) \\ &= \sum_{k=1}^{3} \sum_{m=1}^{3} c_{ki} c_{mj} G_{km} \end{aligned} \tag{112}$$

This can be expressed in matrix notation as $\tilde{G} = C^T G C$.

2.17 HexShells

Hexshells are provided to give the analyst an element with performance similar to a standard shell, but with the mesh topography of a brick. Thus, thin regions of the model can be meshed with hexshells, without concern for the bad aspect ratio of the elements, and with topography consistent with a solid mesh.

The element is documented extensively in the description by Carlos Felippa (see reference 11). The paragraphs in this document summarize the limitations of the shells and the possible usage.

Because hexshells have an inherent thickness direction, it is important to be able to identify that direction. There are (at least) four methods to accomplish this.

natural The *natural* ordering of the nodes in the element can determine the thickness direction. This is the method used by Carlos in developing the element. I believe that the connectivity for the element will indeed have to be modified to properly interface to his software.

sideset The placement of a sideset on one (or both) thickness faces of the elements uniquely identifies the thickness direction.

topology Usually the topology can be used to identify the thickness direction. The hexshell should be used in a sheet. If the hexshells are considered alone, only the free surfaces of the sheet are candidates for the thickness direction. Further, once the thickness direction is established for one element, it must propagate to the neighbors. (Note that this implies that we can't have a self intersecting sheet).

projection The thickness direction could be determined by the closest projection to a coordinate direction.

We will try to support all of the above methods. The *topology* method puts the least burden on the analyst. It is the least explicit however, and the most work to implement (especially in parallel). The next simplest (for the analyst) is the *projection* method. Sideset methods are burdensome for both the analyst and the code development team. The *natural* method is the easiest to implement, but can be next to impossible for the analyst to use.

Input will be structured as follows. Keywords are associated with each method. Only one of the four keywords above can be entered. If no keyword is entered, then *topology* is assumed.

```
Block 9
   HexShell
   orientation sideset='1,2'
   material=9
end
```

or,

```
Block 10
   HexShell
   orientation topology
   material=9
end
```

3 Linear Algebra Issues

3.1 Solution Spaces

There are number of different dimensions in Salinas. These will be summarized here with a focus on using the data within the matlab framework. Examples of how to convert data from one dimensionality to another will be given.

The subject of matrix dimensions is an important one. Salinas has a fairly simple set of dimensions compared to more complex systems like Nastran. However, it is critical that these be well understood if we wish to manipulate the data.

As an example, I consider an eigen analysis of a structure with 9938 nodes. This structure is made of shells and solids. There are no boundary conditions, but there are 9 mpcs applied. I look at only the serial file sizes.

To get the required maps and other m-files, we must select 'mfiles' in the output section. To get the eigenvector data, we must also write the exodus file with 'disp' selected in the output section.
For this model, we have the following important dimensions.

1. #nodes=9938
2. external set= #nodes * 6 dofs/node = 59628
3. `G-set` = # active dofs before boundary conditions = 42708
4. `A-set` = analysis set = # equations to be solved = 42699
5. reduced external set = #nodes * 3 = 29814

There are 3 dofs/node for solid elements, but shells and beams have 6. In aggregate, the total dofs is 42708 before boundary conditions and mpcs are applied. There are no BCs in the model, but there are 9 MPC equations, each of which eliminates 1 dof, so the Aset is reduced to 42699.

Unfortunately, the `eigen_disp*.m` files are written in the reduced external set since this is what the analysts typically want. The bad news is that these m-files are useless to us. The good news is that all the data is available in either `m-files` or in the `exodus` output.

The matrices `Mssr` and `Kssr` contain the mass and stiffness matrices in the `A-set`. They are symmetric matrices and only one half of the off diagonal is stored. To get the complete matrix within `matlab`,

```
>>> K = Kssr + Kssr' - speye(size(Kssr)).*Kssr;
```

The full eigenvectors (in the external set) are available in the output exodus file. To get them use the `seacas` command `exo2mat`.

```
> exo2mat example-out.exo
```

Within `matlab`, the data can be converted to a properly shaped matrix.

```
>>> load example-out
>>> phi = zeros(nnodes*6,nsteps);
>>> tmp = (0:nnodes-1)*6;
>>> phi(tmp+1,:)=nvar01;
>>> phi(tmp+2,:)=nvar02;
>>> phi(tmp+3,:)=nvar03;
>>> phi(tmp+4,:)=nvar04;
>>> phi(tmp+5,:)=nvar05;
>>> phi(tmp+6,:)=nvar06;
```

We now have phi as a matrix with each column corresponding to an eigenvector. However, phi is dimensioned at 59628 x 10 for this example. We clearly can't multiply phi by K for example - the dimensions don't match. To do this we need a map.

We have two maps in our directory. `FetiMap_a.m` is the map from the external set to the A set. Thus we can reduce `phi` to the `A-set` by combining it with `Fetimap_a`. If the `G-set` is desired instead of the `A-set`, replace `FetiMap_a` with `FetiMap`.

```
>>> p2=zeros(max(max(FetiMap_a)),nsteps);
>>> for j=1:nnodes*6
>>>  i=FetiMap_a(j);
>>>  if ( i > 0 )
>>>    p2(i,:)=phi(j,:);
>>>  end
>>> end
```

This is slow. A faster, but less straightforward method is shown here.

```
>>> mapp1=FetiMap_a+1;
>>> tmp=zeros(max(max(mapp1)),nsteps);
>>> tmp(mapp1,:)=phi;
>>> p2=tmp(2:max(max(mapp1)),:);
```

Now we can do all the neat things like `p2'*K*p2`.

To get back to the external set, we again use this map. For example, if we have a vector of dimension 42699,

```
>>> x=1:42699';
>>> XX = zeros(59628,1);
>>> for i=1:59628
>>>   if ( FetiMap_a(i)>0 )
>>>     XX(i)=x(FetiMap_a(i));
>>>   end
>>> end
```

Obviously, similar shortcuts can be made to make this more efficient. One that appears to work is shown here.

```
>>> xtmp=[ 0 x'];
>>> X2=xtmp(mapp1);
```

References

[1] Farhat, Crivelli, and Geradin, "Implicit time integration of a class of constrained hybrid formulations - Part I: Spectral stability theory," *CMAME*, **vol. 125**, no. 71-107, 1995, pp. 71–107.

[2] Chung, J. and Hulbert, G., "A Time Integration Algorithm for Structural Dynamics with Improved Numerical Dissipation: The Generalized alpha method," *Journal of Applied Mechanics*, **vol. 60**, pp. 371–375.

[3] Reese, G., Field, R., and Segalman, D. J., "A Tutorial on Design Analysis Using von Mises Stress in Random Vibration Environments," *Shock and Vibration. Digest*, **vol. 32**, no. 6, 2000.

[4] Craig, R. R., *Structural Dynamics: An Introduction to Computer Methods*, John Wiley & Sons, 1981.

[5] Hughes, T. J. R., *The Finite Element Method–Linear Static and Dynamic Finite Element Analysis*, chap. Appendix 3.1, Prentice-Hall, Inc, 1987, p. 170.

[6] Allman, D. J., "A Compatible Triangular Element Including Vertex Rotations for Plane Elasticity Problems," *Computers and Structures*, **vol. 19**, no. 1-2, 1996, pp. 1–8.

[7] Cook, R. D. and D. S. Malkaus, M. E. P., *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, third edn., 1989.

[8] Batoz, J.-L., Bathe, K.-J., and Ho, L.-W., "A Study of Three-Node Triangular Plate Bending Elements," *International Journal for Numerical Methods in Engineering*, **vol. 15**, 1980, pp. 1771–1812.

[9] Zienkiewicz, O. C. and Taylor, R. L., *The Finite Element Method*, vol. 2, chap. 1, McGraw-Hill Book Company Limited, fourth edn., 1991, pp. 23–26.

[10] Ertas, A., Krafcik, J. T., and Ekwaro-Osire, S., "Explicit Formulation of an Anisotropic Allman/DKT 3-Node Thin Triangular Flat Shell Elements," *Composite Material Technololgy*, **vol. 37**, 1991, pp. 249–255.

[11] Felippa, C. A., "The SS8 Solid-Shell Element: Formulation and a Mathematica Implementation," Tech. Rep. CU-CAS-02-03, Univ. Colo. at Boulder, 2002.

Index